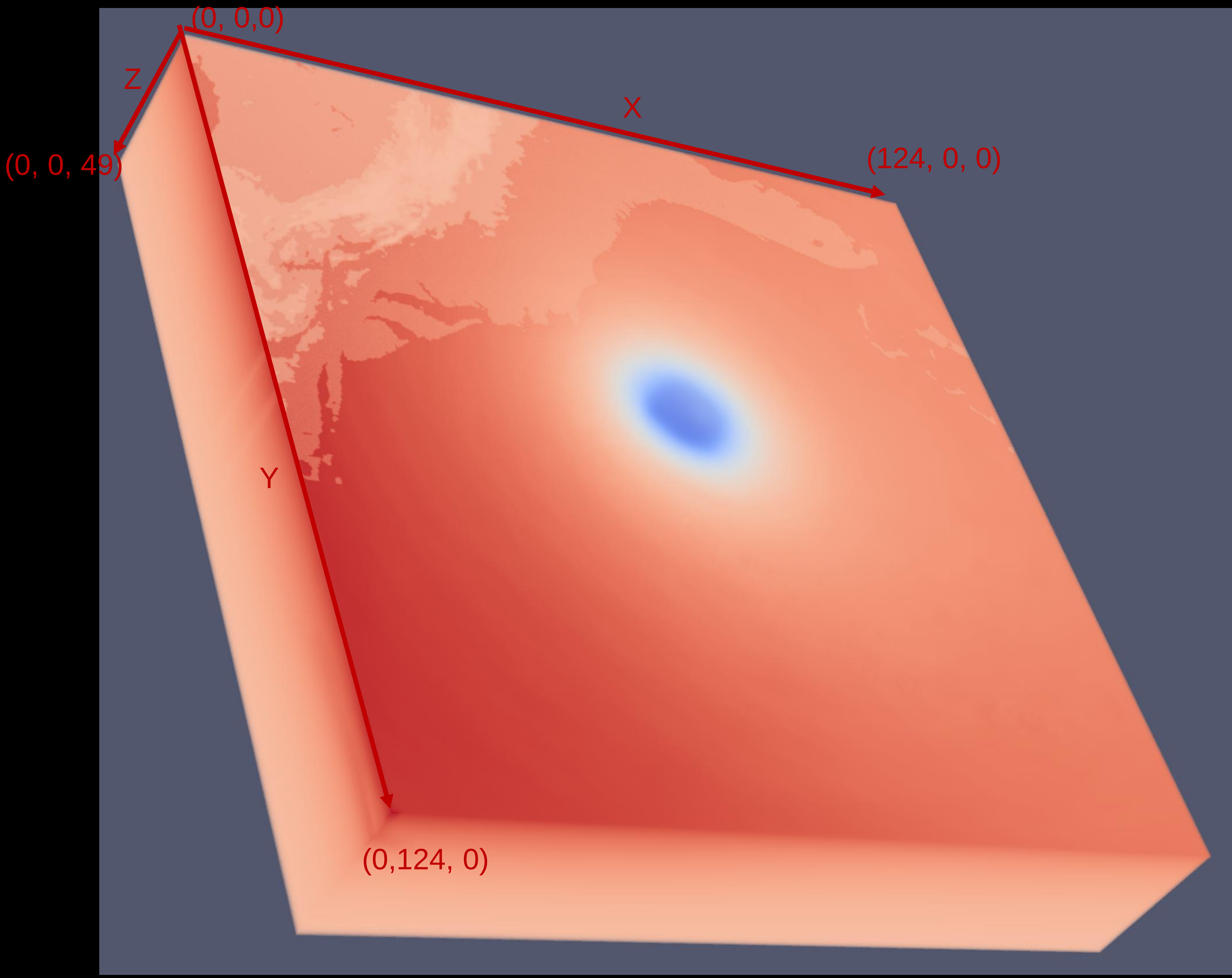
Direct Volume Rendering (3D)

(Hurricane pressure dataset)

This is a 125x125x50 3D dataset

Data value is the pressure

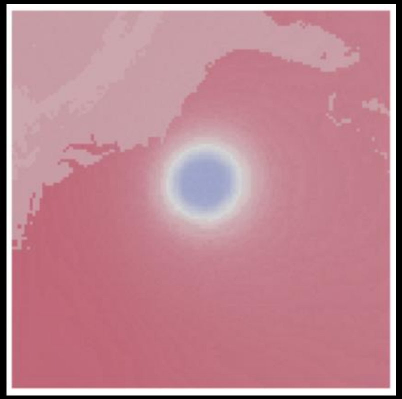


Files

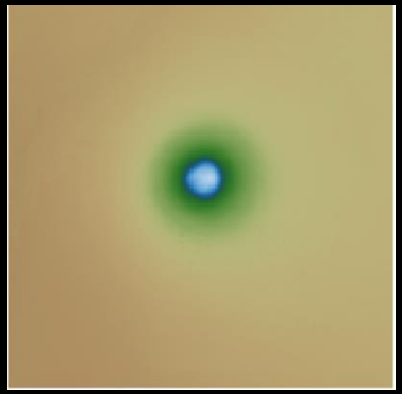
- DVR.ipynb: code template and you should complete this homework in this file and submit this file

- data.npy: the 3D data set (you need it in the working folder)

- TF1.json and TF2.json: two transfer functions for test (you should at least test your implementation on these two transfer functions)

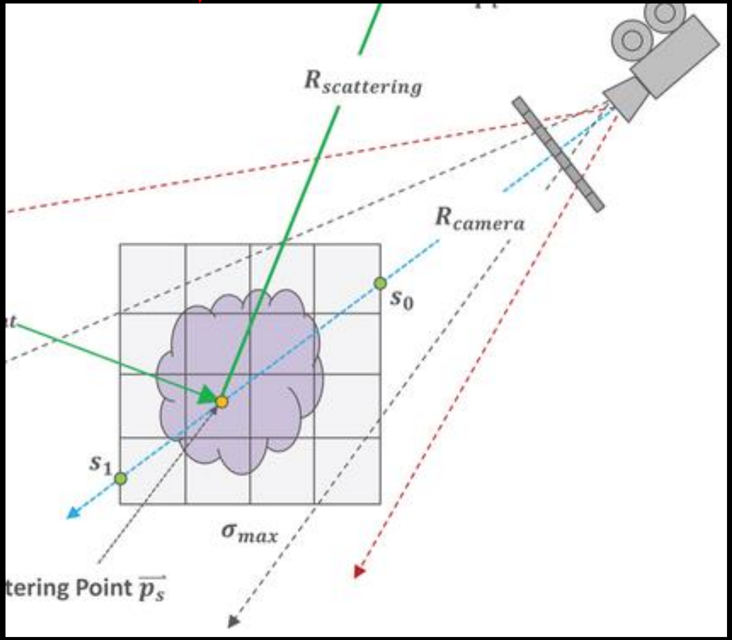


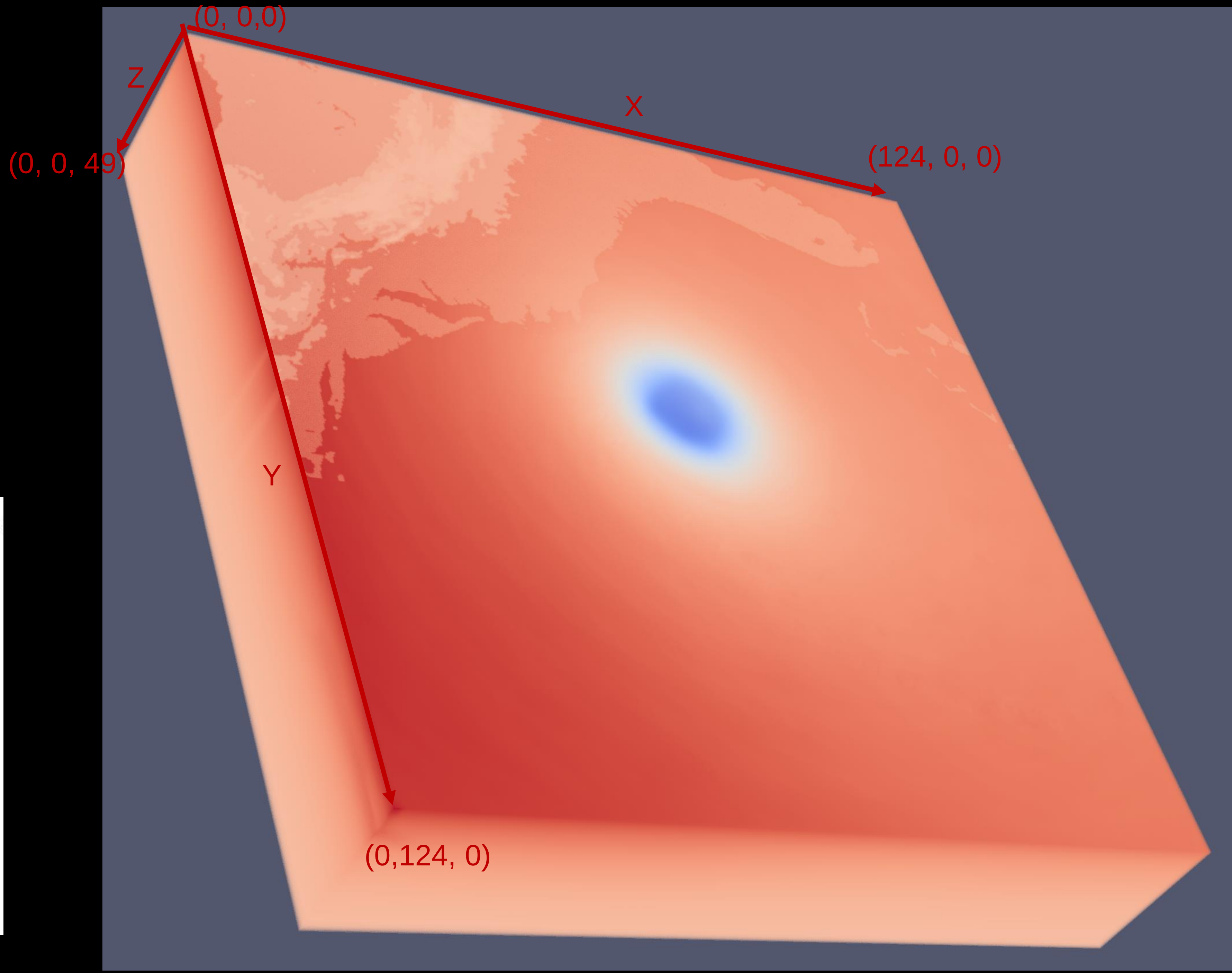
The result of TF1.json



The result of TF2.json

To implement DVR, we should implement a camera model and ray casting algorithm, BUT…….





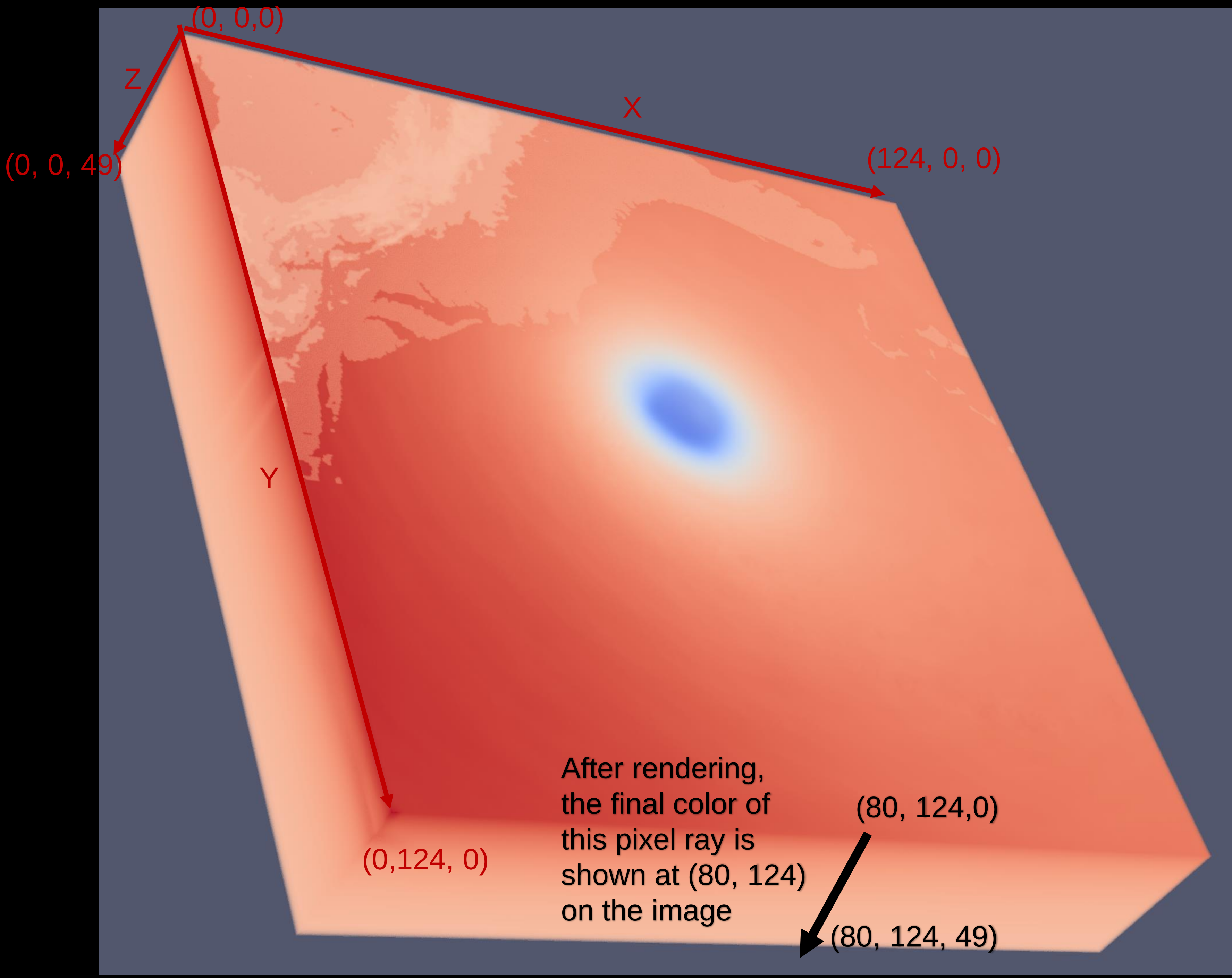
But, to simplify this homework, we won’t do that.

We simply consider points with same x and y coordinates are projected to the same pixel.

And, we render an image with 125x125 resolution. We only cast ray at x and y coordinates with integer number.

The camera is set at the side with z=0 and the casting ray travel to z=49.

We also regularly sample 50 samples on a pixel ray. That means we only get samples to render at z=0, 1, 2, 3, 4…., 49



TF1.json

Transfer Function

- In the code template, we provide a function to load the transfer function file and parse then into arrays (“opacityTransferFunc” and “colorTransferFunc”) for you
- I actually export the transfer function file from Paraview. If you want, you can also export your own to try
- To correctly use the transfer function for volume rendering, you have to implement piece-wise linear interpolation to look up the opacity value and color of a data value

```
[
    {
        "ColorSpace" : "Diverging",
        "Name" : "Preset3",
        "Points" :
        [
            -5395.6650390625,
            0.0,
            0.5,
            0.0,
            -2769.017578125,
            0.023076923564076424,
            0.5,
            0.0,
            -12.338045120239258,
            0.84358978271484375,
            0.5,
            0.0,
            1001.9119873046875,
            0.048717949539422989,
            0.5,
            0.0,
            2926.386474609375,
            0.0,
            0.5,
            0.0
        ],
        "RGBPoints" :
        [
            -5395.6650390625,
            0.23137254902000001,
            0.298039215686,
            0.75294117647100001,
            -844.543212890625,
            0.86499999999999999,
            0.86499999999999999,
            0.86499999999999999,
            2926.386474609375,
            0.70588235294099999,
            0.015686274509800001,
            0.149019607843
        ]
    }
]
```

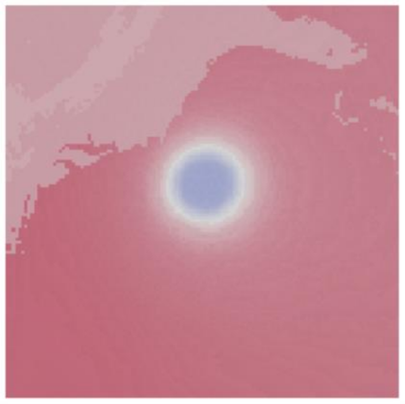
opacity: 5 control points

one control points: Data value, opacity, Useless here, Useless here

color: 3 control points

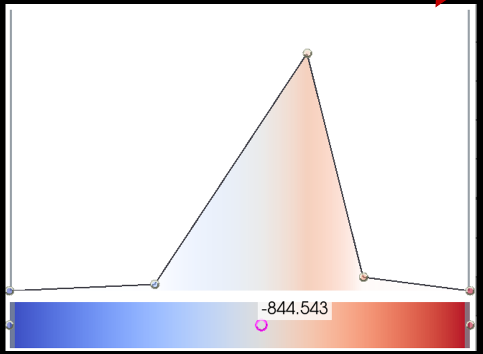
one control points: Data value, R, G, B

Rendered image by TF1.json



Opacity: 5 control points

Color: 3 control points



Template

Initialize(): it load the dataset and transfer function. The input argument is the filename of the transfer function.

```python
import numpy as np
import matplotlib.pyplot as plt
import json

data = 0

# after loading the transfer function,
# each subarray (control point) in an opacity function: [dataValue, opacity]
# each subarray (control point) in a color function: [dataValue, R(0-1), G(0-1), B(0-1)]
opacityTransferFunc = []
colorTransferFunc = []

##### data loading and setup/plot image
##### DO NOT modify this function
def Initialize(tfFileName):
    global data
    plt.rcParams['figure.figsize'] = [5, 5]
    plt.axis('off')
    data = np.load('data.npy')

    f = open(tfFileName)
    jsn = json.load(f)
    jsn = jsn[0]

    opacityTransferFunc.clear()
    colorTransferFunc.clear()
    for i in range( 0, len(jsn['Points']), 4 ):
        tmp = []
        tmp.append(jsn['Points'][i+0])
        tmp.append(jsn['Points'][i+1])
        opacityTransferFunc.append(tmp)
    for i in range( 0, len(jsn['RGBPoints']), 4 ):
        tmp = []
        tmp.append(jsn['RGBPoints'][i+0])
        tmp.append(jsn['RGBPoints'][i+1])
        tmp.append(jsn['RGBPoints'][i+2])
        tmp.append(jsn['RGBPoints'][i+3])
        colorTransferFunc.append(tmp)

###### get data value: x and y are locaion on the image plane, z is coordinate along the pixel depth direction
###### In this data, x index: [0, 125), y index: [0, 125), z index: [0, 49)
def getValue( x, y, z ):
    global data
    return data[ z, x, y ]

###########main

### initialize and load a transfer function, the input argument is the trasnfer function file name
### after loading the opacity function and color function are stored in 'opacityTransferFunc' and 'colorTransferFunc'
Initialize('TF1.json')

##### 'img' is used to store the final image
img = np.zeros([125, 125, 3])

####### implment you direct volume rendering here and store the final image in "img"
print("TODO: you shoud implment direct volume rendering here")


####### show final image (img)
plt.imshow(img)
plt.show()
```

Template

getValue(): the input argument are x, y, z position (integer only) and it will return the value at the location to you

```python
import numpy as np
import matplotlib.pyplot as plt
import json

data = 0

# after loading the transfer function,
# each subarray (control point) in an opacity function: [dataValue, opacity]
# each subarray (control point) in a color function: [dataValue, R(0-1), G(0-1), B(0-1)]
opacityTransferFunc = []
colorTransferFunc = []

##### data loading and setup/plot image
##### DO NOT modify this function
def Initialize(tfFileName):
    global data
    plt.rcParams['figure.figsize'] = [5, 5]
    plt.axis('off')
    data = np.load('data.npy')

    f = open(tfFileName)
    jsn = json.load(f)
    jsn = jsn[0]

    opacityTransferFunc.clear()
    colorTransferFunc.clear()
    for i in range( 0, len(jsn['Points']), 4 ):
        tmp = []
        tmp.append(jsn['Points'][i+0])
        tmp.append(jsn['Points'][i+1])
        opacityTransferFunc.append(tmp)
    for i in range( 0, len(jsn['RGBPoints']), 4 ):
        tmp = []
        tmp.append(jsn['RGBPoints'][i+0])
        tmp.append(jsn['RGBPoints'][i+1])
        tmp.append(jsn['RGBPoints'][i+2])
        tmp.append(jsn['RGBPoints'][i+3])
        colorTransferFunc.append(tmp)

###### get data value: x and y are locaion on the image plane, z is coordinate along the pixel depth direction
###### In this data, x index: [0, 125), y index: [0, 125), z index: [0, 49)
def getValue( x, y, z ):
    global data
    return data[ z, x, y ]

###########main

### initialize and load a transfer function, the input argument is the trasnfer function file name
### after loading the opacity function and color function are stored in 'opacityTransferFunc' and 'colorTransferFunc'
Initialize('TF1.json')

##### 'img' is used to store the final image
img = np.zeros([125, 125, 3])

####### implment you direct volume rendering here and store the final image in "img"
print("TODO: you shoud implment direct volume rendering here")


####### show final image (img)
plt.imshow(img)
plt.show()
```

Template

The output image should be 125x125. You can put the final image into 'img' and 'img' will be shown

```python
import numpy as np
import matplotlib.pyplot as plt
import json

data = 0

# after loading the transfer function,
# each subarray (control point) in an opacity function: [dataValue, opacity]
# each subarray (control point) in a color function: [dataValue, R(0-1), G(0-1), B(0-1)]
opacityTransferFunc = []
colorTransferFunc = []

##### data loading and setup/plot image
##### DO NOT modify this function
def Initialize(tfFileName):
    global data
    plt.rcParams['figure.figsize'] = [5, 5]
    plt.axis('off')
    data = np.load('data.npy')

    f = open(tfFileName)
    jsn = json.load(f)
    jsn = jsn[0]

    opacityTransferFunc.clear()
    colorTransferFunc.clear()
    for i in range( 0, len(jsn['Points']), 4 ):
        tmp = []
        tmp.append(jsn['Points'][i+0])
        tmp.append(jsn['Points'][i+1])
        opacityTransferFunc.append(tmp)
    for i in range( 0, len(jsn['RGBPoints']), 4 ):
        tmp = []
        tmp.append(jsn['RGBPoints'][i+0])
        tmp.append(jsn['RGBPoints'][i+1])
        tmp.append(jsn['RGBPoints'][i+2])
        tmp.append(jsn['RGBPoints'][i+3])
        colorTransferFunc.append(tmp)

###### get data value: x and y are locaion on the image plane, z is coordinate along the pixel depth direction
###### In this data, x index: [0, 125), y index: [0, 125), z index: [0, 49)
def getValue( x, y, z ):
    global data
    return data[ z, x, y ]

###########main

### initialize and load a transfer function, the input argument is the trasnfer function file name
### after loading the opacity function and color function are stored in 'opacityTransferFunc' and 'colorTransferFunc'
Initialize('TF1.json')

##### 'img' is used to store the final image
img = np.zeros([125, 125, 3])

####### implment you direct volume rendering here and store the final image in "img"
print("TODO: you shoud implment direct volume rendering here")


####### show final image (img)
plt.imshow(img)
plt.show()
```

Template

You can implement the direct volume rendering here.

Btw, the rendering process could be slow. That is fine.

```python
import numpy as np
import matplotlib.pyplot as plt
import json

data = 0

# after loading the transfer function,
# each subarray (control point) in an opacity function: [dataValue, opacity]
# each subarray (control point) in a color function: [dataValue, R(0-1), G(0-1), B(0-1)]
opacityTransferFunc = []
colorTransferFunc = []

##### data loading and setup/plot image
##### DO NOT modify this function
def Initialize(tfFileName):
    global data
    plt.rcParams['figure.figsize'] = [5, 5]
    plt.axis('off')
    data = np.load('data.npy')

    f = open(tfFileName)
    jsn = json.load(f)
    jsn = jsn[0]

    opacityTransferFunc.clear()
    colorTransferFunc.clear()
    for i in range( 0, len(jsn['Points']), 4 ):
        tmp = []
        tmp.append(jsn['Points'][i+0])
        tmp.append(jsn['Points'][i+1])
        opacityTransferFunc.append(tmp)
    for i in range( 0, len(jsn['RGBPoints']), 4 ):
        tmp = []
        tmp.append(jsn['RGBPoints'][i+0])
        tmp.append(jsn['RGBPoints'][i+1])
        tmp.append(jsn['RGBPoints'][i+2])
        tmp.append(jsn['RGBPoints'][i+3])
        colorTransferFunc.append(tmp)

###### get data value: x and y are locaion on the image plane, z is coordinate along the pixel depth direction
###### In this data, x index: [0, 125), y index: [0, 125), z index: [0, 49)
def getValue( x, y, z ):
    global data
    return data[ z, x, y ]

###########main

### initialize and load a transfer function, the input argument is the trasnfer function file name
### after loading the opacity function and color function are stored in 'opacityTransferFunc' and 'colorTransferFunc'
Initialize('TF1.json')

##### 'img' is used to store the final image
img = np.zeros([125, 125, 3])

####### implment you direct volume rendering here and store the final image in "img"
print("TODO: you shoud implment direct volume rendering here")

####### show final image (img)
plt.imshow(img)
plt.show()
```